FAULT-TOLERANT PARALLEL ALGORITHMS FOR ADAPTIVE MATCHED-FIELD PROCESSING ON DISTRIBUTED ARRAY SYSTEMS

KILSEOK CHO, ALAN D. GEORGE, AND RAJ SUBRAMANIYAN

High-performance Computing and Simulation (HCS) Research Laboratory

Department of Electrical and Computer Engineering, University of Florida

P.O. Box 116200, Gainesville, FL 32611-6200

Received 14 January 2004

Revised 24 September 2004

Continuous innovations in adaptive matched-field processing (MFP) algorithms have presented significant increases in computational complexity and resource requirements that make development and use of advanced parallel processing techniques imperative. In real-time sonar systems operating in severe underwater environments, there is a high likelihood of some part of systems exhibiting defective behavior, resulting in loss of critical network, processor, and sensor elements, and degradation in beam power pattern. Such real-time sonar systems require high reliability to overcome these challenging problems. In this paper, efficient fault-tolerant parallel algorithms based on coarse-grained domain decomposition methods are developed in order to meet real-time and reliability requirements on distributed array systems in the presence of processor and sensor element failures. The performance of the fault-tolerant parallel algorithms is experimentally analyzed in terms of beamforming performance, computation time, speedup, and parallel efficiency on a distributed testbed. The performance results demonstrate that these fault-tolerant parallel algorithms can provide real-time, scalable, lightweight, and fault-tolerant implementations for adaptive MFP algorithms on distributed array systems.

1. Introduction

Over the past decade, the rapid improvement of signal processing algorithms and better understanding of signal and ocean environment models have resulted in the development of advanced beamforming algorithms for highly cluttered environments. In particular, matched-field processing (MFP) is the process of cross-correlation of a measured field with a replica field derived from the spatial point source response of the medium[19]. The MFP algorithm localizes acoustic sources in range, depth. and azimuth more precisely than plane-wave beamforming methods by using a full-wave acoustic propagation model instead of a simple plain-wave acoustic propagation model for the ocean[20]. Continuous innovations in adaptive MFP algorithms have presented a significant increase in computational complexity that makes development and use of high-performance processing systems imperative in real-world applications[1]. In real-time sonar systems operating with battery power in severe underwater environments, there is a high probability of some part of the system exhibiting faulty behavior at any given time. Specifically, the principal source of failures during missions is node-outage due to battery run-down. Such failures might result in failure of interconnection network, distortion in beam power pattern, and loss of critical processor and sensor elements[2]. As a result, such real-time sonar systems require high dependability to tolerate these failures in typically severe underwater environments. High-performance, efficient and fault-tolerant processing methods are necessary in order to assure high reliability and meet the real-time requirements of distributed sonar array systems in spite of the high probability of processing and sensor failures.

Strategies for fault tolerance can be categorized into algorithm-based, system-level, and application-level methods depending on how and where fault tolerance capabilities are applied to systems. Algorithm-

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **24 SEP 2004**	2. REPORT TYPE
3. DATES COVERED **00-00-2004 to 00-00-2004**	
4. TITLE AND SUBTITLE **Fault-Tolerant Parallel Algorithms for Adaptive Matched-Field Processing on Distributed Array Systems**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of Florida,Department of Electrical and Computer Engineering,High-performance Computing and Simulation (HCS) Research Laboratory,Gainesville,FL,32611**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited**	
13. SUPPLEMENTARY NOTES **The original document contains color images.**	
14. ABSTRACT	
15. SUBJECT TERMS	

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**18**	

based mechanisms use redundant computations within the algorithms to detect and correct errors caused by permanent and/or transient faults in the hardware[3]. This technique is highly algorithm-dependent and has only been applied to a handful of problems with redundant computations[4]. However, this technique has advantages in terms of low overhead and cost when compared with traditional fault-tolerant techniques. This method is employed for simple applications including checksum matrix operations, matrix inversions, and the QR decomposition method[5]. System-level fault tolerance includes redundancy in all system hardware and software components in which recovery actions are undertaken by the system. This technique involves significant redundancy and recovery time because considerable replication and process checkpointing with system hardware and software is necessary in order to make the system transparent to applications and programmers. However, this method can be easily incorporated into a system with little work on the part of the application developers. An example of system-level fault tolerance for distributed computing environments is MPICH-V[6], an implementation of the message-passing interface (MPI) standard that can tolerate loss of volatile resources using a checkpointing method[21, 22]. Finally, application-level techniques have redundancy and recovery actions embedded within the applications themselves, thereby providing efficient and low overhead in fault tolerance by employing information only available at the application level[7]. This technique allows higher fidelity detection and handling of faults than system-level or algorithm-based techniques. It can reduce substantial computational load and redundancy since replication and architecture-dependent checkpointing can be avoided. The application-level fault-tolerant method has been used for sonar beamforming and radar tracking algorithms[7] and is an attractive alternative to reduce extensive hardware and software redundancy in system-level fault tolerance schemes.

High-performance parallel computing techniques have been employed to reduce intensive computational load and improve dependability in various beamforming applications. Fault tolerance in autonomous acoustic arrays has been studied to examine the effects of sensor failures on overall system performance and derive simple and cost-effective techniques to reduce such effects[2]. Parallel beamforming algorithms have been developed for split-aperture conventional beamforming[8], adaptive minimum variance distortionless response (MVDR) beamforming[9, 10], and MFP[11, 12] algorithms. These parallel algorithms based on control and domain decomposition for several beamforming applications achieved promising performance in terms of scalability and resource requirements on distributed computing systems. However, if there are processing and sensor element failures in distributed sonar array systems, these parallel algorithms do not address fault-tolerant mechanisms to recover from the failures.

In this paper, efficient and lightweight fault-tolerant parallel algorithms for adaptive MFP algorithms are developed to tolerate such failures in distributed sonar array systems. In order to reduce the overhead and redundancy of fault tolerance, application-level fault recovery schemes that take advantage of the information only available at the application level are used. Two robust MFP algorithms[13] are employed to compensate for the impacts of sensor failures, and two new fault-tolerant parallel algorithms are developed respectively. The fault-tolerant parallel algorithms are dynamic frequency decomposition and dynamic section decomposition methods based on domain decomposition and achieve low overhead, minimum redundancy, and fast recovery time in distributed array systems. The fault-tolerant parallel algorithms are experimentally analyzed in terms of computation time, speedup, parallel efficiency, and beamforming performance on a distributed PC cluster.

The rest of paper is organized as follows. Section 2 provides an overview of the robust MFP algorithms. Section 3 discusses the computational tasks and control flow diagrams of the sequential robust MFP algorithms. In Section 4, the configurations and features of the two fault-tolerant parallel algorithms are explained and in Section 5, their computational and communication characteristics are investigated.

Section 6 experimentally analyzes performance results of the fault-tolerant parallel algorithms on a distributed testbed. Finally, Section 7 ends with conclusions and directions for future research.

2. Overview of Robust MFP Algorithms

The minimum variance distortionless response MFP (MVDR-MFP) algorithm adaptively attempts to pass a signal received from the desired point with unity gain and suppresses signals arriving from all the other points[14]. This adaptive MFP algorithm localizes the source more accurately than conventional MFP with moderate levels of signal-to-noise ratio (SNR) and precise knowledge about environments since it provides significant interference rejection and sidelobe suppression. However, MVDR-MFP is sensitive to mismatches such as inaccurate knowledge of the environmental parameters and element failures. In this paper, two robust MFP algorithms namely multiple-constraint MVDR-MFP and reduced-rank MVDR-MFP are used to tolerate sensor element failures in the distributed array systems. The two robust MFP algorithms are overviewed briefly in the following sections.

2.1. Multiple-constraint MVDR-MFP algorithm

Multiple-constraint MVDR matched-field processing (MCMVDR-MFP) calculates an optimal weight vector for the array by using linear multiple constraints. MCMVDR-MFP employs the multiple-constraint, matched-field processor derived by Schmidt et al[15] whose performance is subtle to the choice of the constraints. In this paper, to minimize the impact of element failures, the MCMVDR-MFP algorithm applies additional constraints to MVDR-MFP instead of using a single constraint. These constraints are composed of unity gain for the desired point as in MVDR, and zero gain for all the defective elements. For M given constraints, MCMVDR-MFP calculates the optimal weight vector to minimize output power S_{mcmvdr}, while maintaining M constraints given as

$$S_{mcmvdr} = w^H R\, w \;\; subject\ to \;\; w^H e_1 = d_1,..,w^H e_M = d_M \tag{1}$$

where w is the weight vector of MCMVDR-MFP, R is the cross-spectral matrix (CSM) of the input data for the array, e_i is the constraint vector for the i^{th} constraint point, and d_i is the constraint value for the i^{th} constraint point[15]. Solving Eq. (1) using the Lagrange multiplier method, the weight vector is derived as

$$w_{mcmvdr} = R^{-1} E (E^H R^{-1} E)^{-1} D^H \tag{2}$$

where $E = [e_1 \,|\, e_2 \,|..|\, e_M]$ is the constraint replica matrix and $D = [d_1, d_2,.., d_M]^T$ represents the constraint column vector. Finally, by substituting w_{mcmvdr} from Eq. (2) into Eq. (1), the output power for a steering position (r, z) is obtained as

$$S_{mcmvdr} = D (E^H R^{-1} E)^{-1} D^H \tag{3}$$

where S_{mcmvdr} is the detection factor at range r and depth z that indicates the likeliness of detection for a given data set. In this algorithm, selecting suitable constraints for a given environment and system is essential in order to achieve better performance. The multiple linear constraints for MVDR-MFP are selected to have unity gain for the desired point and zero gain for the defective elements[13].

2.2. Reduced-rank MVDR-MFP algorithm

The reduced-rank MVDR matched-field processing (RRMVDR-MFP) algorithm[16] is used to compensate for element failures by projecting the array output vectors into a lower dimensional subspace while preserving the signal vectors. The CSM is transformed into the eigensubspace and the K subspaces preserving signals out of N ($K \leq N$) are selected by using a subspace selection criteria. Next, the reduced subspace is used to calculate the pseudo-inverse of the CSM, optimal weighting vectors, and finally the beamforming output response. By applying an eigen-decomposition method, the CSM is decomposed into a set of eigenvectors and eigenvalues given as

$$R = E[XX^H] = ULU^H = \sum_{i=1}^{K} \lambda_i u_i u_i^H + \sum_{i=K+1}^{N} \lambda_i u_i u_i^H \tag{4}$$

where $U = [u_1 | u_2 | .. | u_n]$ is the orthogonal matrix whose columns are composed of the eigenvectors u_i associated with λ_i and $L = diag[\lambda_1 \lambda_2 ... \lambda_n]$ is the diagonal matrix whose diagonal elements consist of the eigenvalues λ_i ($\lambda_1 \geq \lambda_2 \geq .. \geq \lambda_n$) which are sorted in decreasing order. As shown in Eq. (4), the CSM is divided into two summation terms of the eigenvectors and eigenvalues where the first term is composed of the signal eigenvectors associated with the K largest eigenvalues and the other consists of the noise eigenvectors corresponding to the (N-K) smallest eigenvalues. When there are faulty elements in the array, the noise vectors associated with small eigenvalues, which disproportionately affect the MVDR output, degrade the spectral output of the MVDR beamformer[23]. RRMVDR-MFP excludes the noise eigenvectors corresponding to the least significant eigenvalues and reduces the rank of CSM from N to K. Thus, the formation of the weight vector for RRMVDR-MFP is calculated as

$$w_{rrmvdr} = \frac{\sum_{i=1}^{K} \frac{1}{\lambda_i} (u_i^H p) u_i}{\sum_{i=1}^{K} \frac{1}{\lambda_i} |u_i^H p|^2} \tag{5}$$

where p is the replica vector calculated from the complex acoustic propagation model for the ocean and K is the rank of the reduced CSM. The final spectral output of RRMVDR-MFP is given in Eq. (6).

$$S_{rrmvdr} = \frac{1}{\sum_{i=1}^{K} \frac{1}{\lambda_i} |u_i^H p|^2} \tag{6}$$

RRMVDR-MFP can achieve enhanced robustness by excluding noise vectors distorted by the element failures and environmental mismatch.

3. Sequential Robust Algorithms

As described in the previous section, two robust algorithms are employed to compensate for the performance degradation presented by sensor element failures. To build baselines for the fault-tolerant parallel algorithms, the two robust algorithms are applied to two dynamic parallel decomposition algorithms discussed in Section 4. In the following sections, sequential MCMVDR and RRMVDR algorithms are presented in detail, respectively.

3.1. *Sequential MCMVDR-MFP algorithm*

The computational tasks of the MCMVDR-MFP algorithm include Fast Fourier Transform (FFT), cross-spectral matrix calculation (CSM), CSM inversion (INV), steering, and broadband averaging as depicted in Fig. 1. The FFT stage transforms the input sample data received by the array of sensors from time domain to frequency domain. The computational complexity of the FFT task implemented by the radix-2 butterfly method is generally $O(L\log L)$ in terms of data length L. However, the FFT stage in MVDR-MFP has a complexity of $O(N)$ in terms of the number of sensor nodes because the FFT process is simply replicated for each sensor node and the FFT data length is fixed.

The CSM stage calculates the CSM of the input data vector in the frequency domain. The computational complexity of this stage is $O(N^2)$ in terms of the number of sensor nodes. The inversion stage calculates the inverse CSM matrix. The computational intensity of matrix inversion increases with the size of matrix and there exist numerous inversion techniques to reduce computation time. In this paper, the Gauss-Jordan elimination (GJE) method[17] is employed to calculate the inverse CSM since it is a widely used technique in beamforming applications with a relatively small memory requirement. The computational complexity of the inversion stage using the GJE method is $O(N^3)$ in terms of size of the matrix.

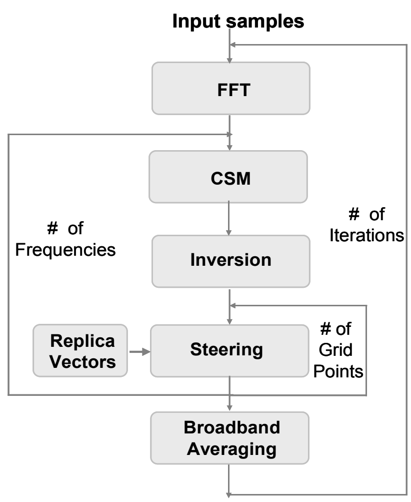


Fig. 1. Computational block diagram of the sequential MCMVDR-MFP algorithm.

The steering stage is responsible for steering an array and calculating the output power for every steering position. The steering stage calculates the output power from the inverse CSM and the replica vectors over the grid points where sources are likely to be present. The complexity of the steering stage for the narrowband MCMVDR-MFP beamformer is $O(RDFN^2)$ because the steering loop has a complexity of $O(FN^2)$ in terms of the number of nodes N and the number of faulty nodes F and is executed as many times as the number of range R and depth D grid points shown in Fig. 1. Along with calculation of output power, the steering stage includes the replica vector generation task that is invariant to input data. The computational procedure for replica vector generation is described in detail by Porter[18]. The replica vector generation task has high-order complexity due to its computationally intensive procedures such as eigenvalue and eigenvector computations. However, as with the FFT stage complexity, the complexity of the replica vector generation is linear with respect to the number of nodes since each sensor node is required to generate the replica vectors for entire range and depth points separately. Despite the linear complexity of the task in terms of N, the computational burden presented by the replica vector generation task is very significant due to a substantial scalar factor. Among all the computational tasks in this experiment, the steering stage is the most computationally intensive. Moreover, the steering stage has a linear complexity in terms of the number of faulty nodes and the computational load of this steering stage is significantly increased as the number of faulty nodes increases.

The broadband averaging stage is required to calculate the average beamformer output for multiple selected frequency bins. Interference signals in the sidelobe are smoothed and signals near the main lobe are enhanced by incoherently averaging the narrowband beamformer outputs over selected frequency bins, because the positions of sidelobes are generally frequency-dependent whereas the location of the main lobe remains constant[24]. The incoherent broadband MVDR-MFP beamformer output is given by

$$S_{av}(r,z) = \frac{1}{B}\sum_{i=1}^{B} S_{mcmvdr}(f_i, r, z) \tag{7}$$

where B is the number of frequency bins selected from the FFT output, f_i represents the i^{th} frequency bin, and $S_{mcmvdr}(f_i, r, z)$ is the narrowband output power of the MCMVDR-MFP algorithm for the i^{th} frequency bin. The narrowband processing including the CSM, inversion, and steering stages for each individual frequency bin is performed as many times as the number of the selected frequency bins for broadband processing. Hence, as the number of frequency bins is increased, the MCMVDR-MFP beamformer has to compute more narrowband beamforming results. This broadband processing vastly increases the computation time of the MCMVDR-MFP algorithm. Finally, the outermost loop including all stages of the MCMVDR-MFP algorithm is performed once every iteration using one snapshot of input data from an array of sensors.

3.2. Sequential RRMVDR-MFP algorithm

The computational tasks of the RRMVDR-MFP algorithm include Fast Fourier Transform (FFT), CSM, eigen-decomposition (ED), steering, and broadband averaging. In RRMVDR-MFP, the FFT, CSM, and broadband averaging computational stages are the same as in MCMVDR-MFP albeit with S_{rrmvdr} in the latter stage instead of S_{mcmvdr}. Fig. 2 shows the computational block diagram of RRMVDR-MFP. The computational flow diagram of this algorithm is the same as that of the MCMVDR-MFP algorithm described in Section 3.1 except for the eigen-decomposition stage. The eigen-decomposition stage transforms the CSM into the eigenvalue and eigenvector subspaces. The reduced subspace preserving signal information is selected by using subspace selection criteria and is used to calculate the pseudo-inverse of the CSM instead of using a matrix inversion method. The Jacobi transformation method[17] is used to transform the CSM into the eigensubspace and involves a computational complexity of $O(N^3)$ with regard to size of the matrix. The steering stage calculates the output power of narrowband frequency bins by using the reduced subspace as in Eq. (6). The steering stage is performed as many times as the number of grid points to steer an array for all possible locations of sources under consideration. The average output power for the selected frequency set is obtained in the broadband averaging stage. The CSM, ED, and steering stages are repeated for the number of selected frequency bins for broadband processing.

4. Fault-tolerant parallel Algorithms for Robust MFP Algorithms

As described in the previous sections, the robust MVDR-MFP algorithms with a large number of sensor nodes, a wide range of frequency bins, and dense grid points require both significant computational complexity and high dependability. Fault-tolerant parallel processing can be a lightweight solution to execute the robust MFP algorithms in real-time and to provide high reliability on distributed array systems. The fault-tolerant parallel algorithms are designed to tolerate node failures on distributed array systems, which consist of smart processing nodes connected through communication networks. In this paper, four fault-tolerant parallel algorithms are developed with the two robust MFP algorithms using coarse-grained domain decomposition methods to exploit more concurrency, dependability, and lower communication overhead. The following sections describe the two fault-tolerant parallel algorithms: dynamic frequency decomposition and dynamic section decomposition.

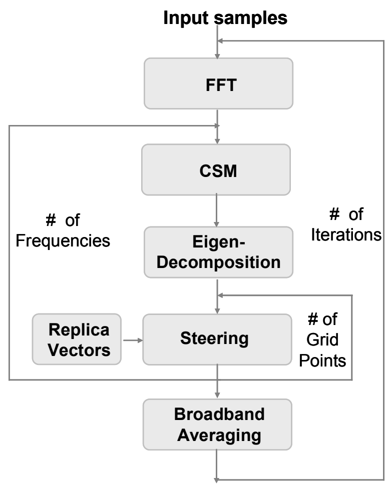


Fig. 2. Computational block diagram of the sequential RRMVDR-MFP algorithm.

4.1. Dynamic frequency decomposition algorithm

The two robust MFP algorithms have concurrency between beamforming operations of different frequency bins since processing of one frequency bin can be executed in parallel with another frequency bin. Dynamic frequency decomposition (DFD) is a dynamic decomposition method extended from the frequency decomposition (FD) algorithm[12] which statically allocates tasks for different subsets of frequency bins to distinct processors. When there are node failures resulting in workload imbalance between nodes, DFD dynamically reallocates processing for different subsets of frequency bins within a given beamforming iteration to non-faulty nodes. DFD is an input domain decomposition technique that exploits data parallelism between processing tasks of different frequency bins. In the presence of faulty nodes, the CSM data assigned to the faulty nodes must be preserved for recovery. A checkpointing method that stores the CSM data of each node and rolls back to the previous checkpoint can be used to recover when a node is faulty. Such checkpointing methods present severe communication and recovery time. To reduce such additional communication and recovery time, each node calculates the CSM data for all the frequency bins, stores it in memory, and uses it when needed. Hence, the CSM stage is excluded from the parallel implementation in the DFD algorithm since the sequential fraction of this computationally simpler stage is not expected to cause a substantial computational load.

In a distributed array system, the FFT output data from each node must be transmitted to other processors for a subsequent beamforming stage. Each node is only responsible for performing the partial beamforming output for a subset of narrowband frequency bins, and therefore each node must share its beamforming output of all grid points to calculate the average of the total beamforming output for all broadband frequency bins. DFD requires two all-to-all communications in each node per iteration, where one all-to-all communication is needed to distribute the FFT data and the other to collect the narrowband beamforming output. To lessen the communication load, a data packing method is employed to combine the partial beamforming output of the previous iteration and the FFT data of the current iteration as one data packet. This data packing decreases the communication cost by eliminating the overhead of initiating communications twice per iteration. However, the DFD algorithm still needs a significantly large communication message size due to the partial beamforming results, since the partial beamforming outputs of all grid points on the search space impose a substantially large communication message on each node as in the FD algorithm.

As shown in Fig. 3, each node performs an FFT operation on the input data received from its own sensor for the current iteration. Each node performs data packing for the FFT output data from the current iteration and the partial beamforming output from the previous iteration, and then executes an all-to-all communication for this data packet. Each node obtains the final beamforming output from the previous iteration by averaging the beamforming output for broadband frequencies. In the 3-node array configuration, the FFT data from the current iteration is communicated to all the other nodes as in FD. In this paper, it is assumed that the faulty nodes are detected by the MPI middleware. Thus, the master node emulates the status of faulty nodes by using a fault injection module and sends the status of all nodes to the other nodes through the data packet. If two other nodes detect that a node is defective, the faulty node is removed from this parallel processing. Thus, tasks for all frequency bins are dynamically reassigned to the non-faulty nodes. For instance, if node 1 is faulty, half the frequency bins are dynamically reassigned to node 0 and the other half to node 2. And each node then calculates CSM for all the frequency bins and carries out the inversion and steering stages for its assigned frequency bins to calculate a narrowband beamforming output over all grid points. The narrowband beamforming output is added to the running sum to obtain the partial beamforming outputs for the assigned frequency bins per node. The above beamforming procedures are repeated for the next iteration. As the number of nodes is increased, the message size of each communication per node is fixed since the partial beamforming outputs over all the grid points and all the FFT output data for the selected frequency bins are sent to other nodes.

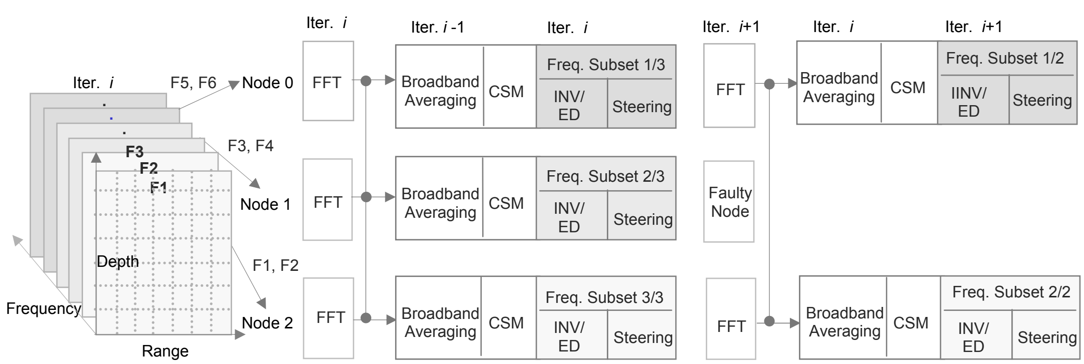


Fig. 3. Block diagram for dynamic frequency decomposition in a 3-node array.

4.2. *Dynamic section decomposition algorithm*

Dynamic section decomposition (DSD) is based on data parallelism between beamforming tasks of different grid points in the two robust MFP algorithms. The DSD algorithm is a dynamic decomposition technique extended from the section decomposition (SD) algorithm[12] which distributes processing of different subsets of grid points within a given beamforming iteration into different processing nodes in the output domain. When there is workload imbalance between nodes in the presence of node failures, DSD dynamically reallocates processing for different grid points into the non-faulty nodes. The DSD algorithm has a sequential dependency between the INV/ED stage and the steering stage because full spectrums of the inverse CSM or the eigensubspace are required for the calculation of the beamforming output for a single grid point. Therefore, the INV/ED stage is excluded from the parallel implementation in the DSD algorithm. This sequential fraction in ED will highly increase the parallel execution time with a large number of nodes in DSD since the ED stage using the Jacobi transformation requires much higher computational complexity than the INV stage even though both stages have the same cubic complexity in terms of the number of nodes. The ED stage requires that the loop having complexity of $O(N^3)$ is repeated on the order of 20 times to achieve convergence in the calculation of eigenvectors and eigenvalues[17]. In the DSD algorithm, each node requires two all-to-all communications as in DFD. To lessen this communication load, the data packing technique is used in the DSD algorithm as well, resulting in one all-to-all communication per iteration. As the number of nodes is increased, the message size of each communication is decreased in the DSD algorithm because the number of beamforming outputs for fixed number of grid points is divided across multiple nodes while the message size is fixed in the DFD algorithm as discussed in the previous section.

The block diagram in Fig. 4 illustrates the DSD method for a 3-node array. In the DSD algorithm, each node performs the FFT task, the data packing operation, and an all-to-all communication between nodes similar to the DFD algorithm. Each node executes the CSM and INV/ED stages to obtain the inverse CSM in MCMVDR-MFP or the eigensubspace of the CSM in RRMVDR-MFP from the current iteration. In the 3-node array configuration, the overall grid points from the current iteration are distributed to the three nodes. The first subset of grid points are allocated to node 1, the second subset of grid points to node 2, and so on. When two nodes detect the third node as faulty, the faulty node is removed from this parallel processing. The beamforming tasks for all grid points are dynamically reassigned to the non-faulty nodes. For instance, if node 2 is faulty, beamforming operations for half the grid points are reassigned to node 0 and those for the other half grid points to node 1. For its assigned subset of grid points, each node performs the steering task to obtain partial beamforming outputs for all the selected frequency bins and then calculates the average of the beamforming outputs of broadband frequency bins. The above beamforming processes are repeated for the next iteration. In the DSD algorithm, it is expected that the communication latency will be lower than that of the DFD algorithm because the message size for each communication is decreased with increased system size.

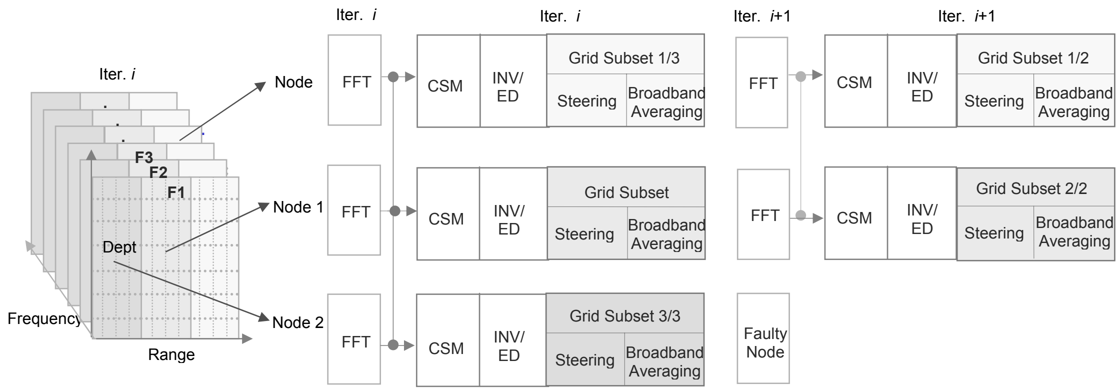


Fig. 4. Block diagram for dynamic section decomposition in a 3-node array.

5. Computational Complexity and Communication

In this section, we analyze computational complexity, communication pattern and message size for the fault-tolerant parallel algorithms described in Section 4. The computational complexities of the two robust algorithms are shown in Table 1 where N is the number of nodes, B is the number of frequency bins, R is the number of range points, D is the number of depth points, F is the number of faulty nodes, and K is the rank of the reduced CSM. When broadband averaging is implemented, the broadband averaging is partially calculated in the steering stage to reduce necessary memory, and its execution time is relatively small compared to other stages. Thus, the complexity and execution time of the broadband averaging is included in that of the steering stage.

As described in Section 3, in the sequential robust algorithms with a fixed number of data points per sensor, the FFT stage for all sensor nodes requires a computational complexity of $O(N)$. The CSM stage involves a complexity of $O(BN^2)$, and the INV/ED stage has a complexity of $O(BN^3)$. The steering stage of the MCVMDR-and RRMVDR-MFP algorithms has a complexity of $O(BRDFN^2)$ and $O(BRDNK)$, respectively. The computational complexities of all the stages in the two fault-tolerant parallel algorithms except the CSM and INV/ED stages are reduced by a factor of N compared to the sequential algorithms. This behavior is because all the parallel algorithms distribute their computational loads into N nodes. However, the computational complexities of the CSM stage in DFD, and the CSM and INV/ED stages in DSD, are not decreased since these stages are excluded from the parallel implementation due to their dependency between computational tasks. It should be noticed that as the number of faulty nodes increases, the computational complexity of MCMVDR-MFP is linearly increased but that of RRMVDR-MFP is not[13].

Table 1. Computational complexities of sequential and fault-tolerant parallel algorithms.

	MCMVDR			RRMVDR		
	Sequential	DFD	DSD	Sequential	DFD	DSD
FFT	$O(N)$	$O(1)$	$O(1)$	$O(N)$	$O(1)$	$O(1)$
CSM	$O(BN^2)$	$O(BN^2)$	$O(BN^2)$	$O(BN^2)$	$O(BN^2)$	$O(BN^2)$
Inversion/Eigen-decomposition	$O(BN^3)$	$O(BN^2)$	$O(BN^3)$	$O(BN^3)$	$O(BN^2)$	$O(BN^3)$
Steering	$O(BRDFN^2)$	$O(BRDFN)$	$O(BRDFN)$	$O(BRDNK)$	$O(BRDK)$	$O(BRDK)$

The communication patterns and message sizes of the two parallel algorithms are shown in Table 2. Here the number of frequency bins used is 64, the number of grid range and depth points is 40 and 80, respectively, and the number of sensor nodes is N. Each unicast communication message is composed of the status data of all the nodes, the FFT data, and the partial beamforming output data. An all-to-all communication scheme that requires $N(N-1)$ *send/receive* unicast communications is used for both DFD and DSD algorithms. As the number of nodes is increased, the message size of each communication is fixed for the DFD algorithm, but that of the DSD algorithm decreases as shown in Table 2. From these communication characteristics, it will be expected that the DFD algorithm has higher communication load than the DSD algorithm.

6. Experiment and Performance Analysis

The performance of the fault-tolerant parallel algorithms is experimentally analyzed on a distributed platform which has multiple processing units connected by loosely coupled communication links. The testbed is a Linux-based cluster of 32 PCs where each node consists of a 1.33 GHz AMD Athlon processor with 256 MB of memory. The interconnection fabric between computers is 100 Mbps switched Fast Ethernet. The MPICH-1.2.5 middleware is employed to communicate and synchronize between processors in this distributed cluster.

The system and problem parameters used in these experiments are 64 frequency bins, 40 grid points in range, 80 grid points in depth, and up to 32 nodes. In this section, parallel performance factors are analyzed in terms of sequential and parallel execution times, speedup, and parallel efficiency to demonstrate the effects of the fault-tolerant parallel algorithms presented in the previous sections. In these experiments, the execution and communication times for one beamforming iteration are measured in terms of the average number of CPU clock cycles instead of using the wall clock time.

Table 2. Communication pattern and message size of fault-tolerant parallel algorithms.

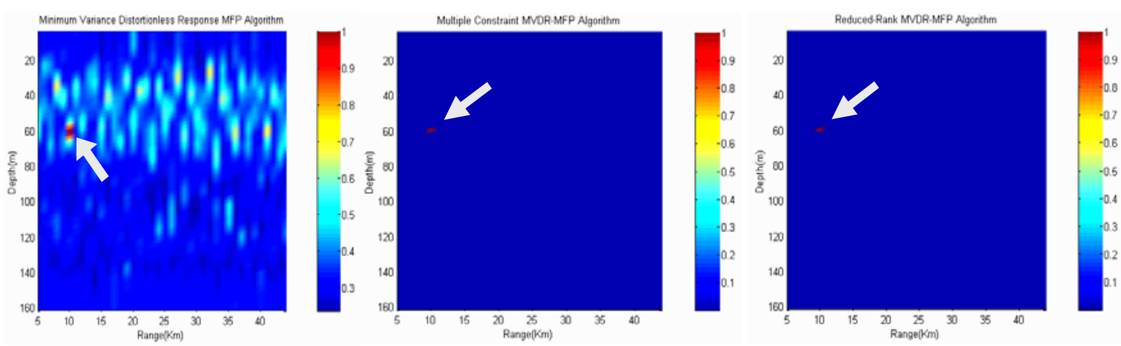
	DFD	DSD
Pattern	All-to-all communication	All-to-all communication
Number of communications per iteration	$N(N-1)$ send/receive communications	$N(N-1)$ send/receive communications
Message size per communication	8 + (# of frequency bins × 2 + # of grid points) × # of bytes per double (8) = 26632 bytes	8 + (# of frequency bins × 2 × 8) + (# grid points × 8 ÷ # of nodes) = 1032 + (25600 ÷ N) bytes

6.1. Beamforming performance

The two robust MFP algorithms are implemented with the fault-tolerant parallel algorithms in the distributed PC cluster and the ambiguity surfaces are investigated to verify if they localize a source position exactly in the presence of node failures. For a given set of environmental and system parameters, the final beamforming output is commonly depicted as an ambiguity surface. The pressure field data was generated by the Kraken normal mode model for a point source which is located at 60 m in depth and 10 km in range, and has 200 Hz frequency and 10 propagating modes. The vertical line array contains 32 hydrophones spaced at 3 m apart from 10 m to 103 m in depth. The noise component for each hydrophone is a Gaussian distribution with zero mean and SNR of 10 dB.

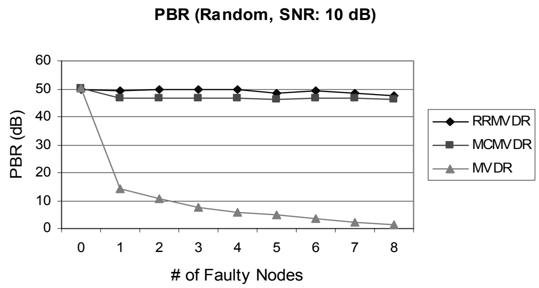
Fig. 5 shows the ambiguity surfaces of MVDR-MFP and the two robust MFP algorithms, where the 16th, 18th, and 20th nodes are defective and SNR is 10 dB. The defective nodes have zero input signals since it is assumed that the array of sensors receives no pressure field from the defective elements. It can be seen that the MVDR-MFP algorithm localizes the source correctly at 60 m in depth and 10 km in range, but generates much higher sidelobe levels and background noises than the two robust algorithms due to the element failures. As shown in Figs. 5b and 5c, the MCMVDR-MFP and RRMVDR-MFP algorithms achieve better localization performance than MVDR-MFP by suppressing sidelobes and background noises caused by the element failures.

In order to provide a means of quantifying characteristics of the ambiguity surface, the peak-to-background ratio is defined as $PBR = (p - \mu)/\mu$ where p is the peak value and μ is the mean background level which is calculated by excluding a small number of points around the peak. PBR is extensively used as a concrete performance metric in beamforming applications[25]. A higher PBR implies that performance of source localization is better. The peak-to-background ratios for random faulty-node configuration were calculated from 50 independent trials of the noise and faulty-node position. PBR of the robust MVDR-MFP algorithms is depicted in Fig. 6a in terms of SNR of 10 dB and up to eight faulty nodes. The faulty nodes are randomly distributed at any position in the array as many as a given number of defective elements. As the number of faulty nodes increases, the PBR of MVDR-MFP rapidly decreases due to the mismatch generated by faulty nodes. However, MCMVDR-MFP and RRMVDR-MFP algorithms achieve better beamforming performance than the MVDR-MFP algorithm for the given defective elements. Fig. 6b illustrates the PBR of these algorithms in terms of the same configurations but in this case with -10 dB SNR. As SNR decreases, the PBR of the robust algorithms decreases as expected. The performance of the robust algorithms with SNR of -10 dB follows almost the same trends as in Fig. 6a.

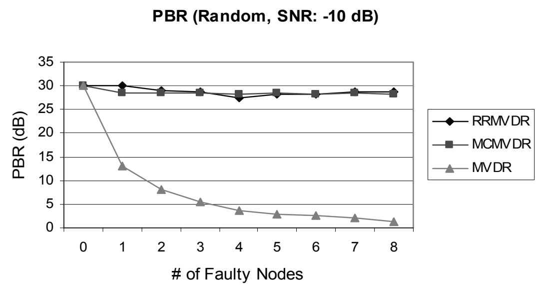


(a) (b) (c)

Fig. 5. Ambiguity surfaces of MVDR-MFP (a), MCMVDR-MFP (b), and RRMVDR-MFP (c) algorithms.



(a)



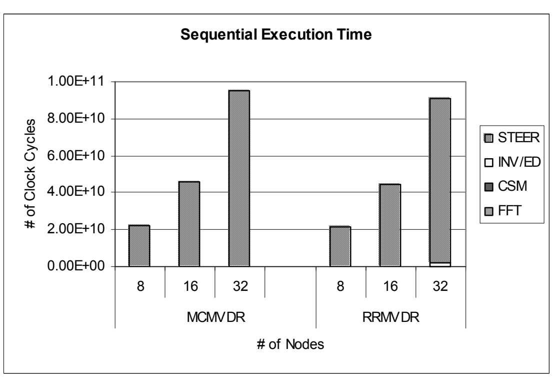
(b)

Fig. 6. Peak-to-background ratios of with SNR of 10 dB (a) and -10 dB (b).

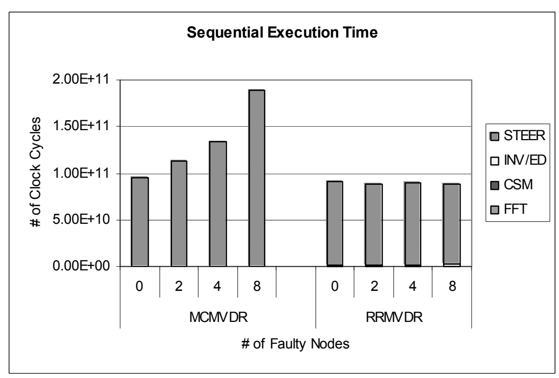
6.2. Sequential execution time

The sequential execution times of the MCMVDR-MFP and RRMVDR-MFP algorithms are measured on a single processing unit in the testbed. Figs. 7a and 7b illustrate the sequential execution times versus the number of sensor nodes and the number of faulty sensor nodes, respectively. In Fig. 7a, the system size varies from eight to 32 nodes and none of the nodes are faulty. In Fig. 7b, the system size is fixed at 32 nodes but the number of faulty nodes varies from zero to eight.

As shown in Fig. 7a, the sequential execution time results demonstrate that the steering stage is the most computationally dominant stage because the complexity of the steering stage is much higher than that of other stages due to the substantial number of grid points. As the number of nodes increases, the sequential execution times in the two robust algorithms is linearly increased since the steering stage includes the replica vector generation task whose computational complexity is linear in terms of the number of nodes as described in Section 3. When there is no faulty node, the sequential execution time of MCMVDR-MFP is slightly higher than that of RRMVDR-MFP since the steering stage of MCMVDR-MFP has a higher computational load in comparison to RRMVDR-MFP as shown in Table 1.



(a)



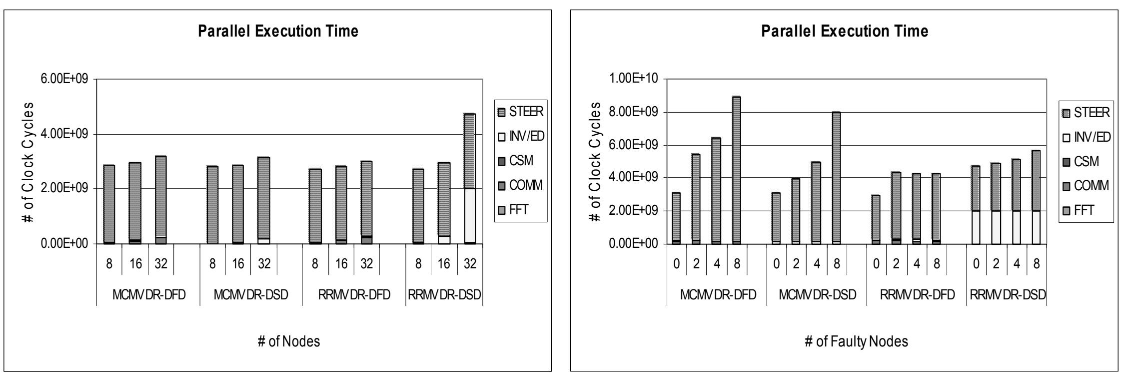
(b)

Fig. 7. Sequential execution time per iteration versus system size (a) and number of faulty nodes (b).

Fig. 7b depicts the sequential execution times of the two robust algorithms in terms of the number of faulty nodes. The steering stage is the most computationally dominant stage as in Fig. 7a. As the number of faulty nodes increases, the sequential execution time of MCMVDR-MFP rapidly increases but that of RRMVDR-MFP does not because computational complexity of the steering stage in MCMVDR-MFP is linearly increased as the number of faulty nodes increases but not in RRMVDR-MFP. From these experimental results, it can be inferred that the RRMVDR-MFP algorithm is computationally more efficient than the MCMVDR-MFP algorithm when there are faulty nodes in the distributed array systems.

6.3. Parallel execution time

Figs. 8a and 8b illustrate the parallel execution times of all four combinations of parallel and robust algorithms with the same system configurations as in Fig. 7. Of course, here the nodes perform both sensing and processing. As shown in Fig. 8a, the execution times of all the fault-tolerant parallel algorithms are only slowly increased as system size increases with no faulty nodes since computational complexities of these parallel algorithms are reduced by a factor of N in the steering stage when compared to the sequential algorithms. The DFD algorithm requires a larger message size than the DSD algorithm. As a result, communication time with the DFD algorithm increases with increase in system size on the PC cluster. The steering stage is still the only computationally dominant stage except in RRMVDR-DSD as was shown in Fig. 7a. As the number of nodes increases, the parallel execution time of the ED stage in RRMVDR-DSD is rapidly increased since the sequential bottleneck of this stage highly increases the parallel execution time as discussed in Section 4.



(a) (b)

Fig. 8. Parallel execution time per iteration versus system size (a) and number of faulty nodes (b).

Fig. 8b shows the parallel execution times of fault-tolerant parallel algorithms where the number of faulty nodes varies from zero to eight and the number of nodes is fixed at 32. The parallel execution times of MCMVDR-DFD are largely increased as the number of faulty nodes increases because the computational complexity of MCMVDR is linearly dependent on the number of faulty nodes as shown in Table 1 and there is an imbalanced workload distribution in DFD. In particular, as the number of faulty nodes increases from zero to two, execution time of DFD is rapidly increased. When workloads of the faulty nodes are reassigned to non-faulty nodes, there is a workload imbalance between non-faulty nodes because the number of frequency bins is not large enough to distribute their beamforming tasks into all non-faulty nodes evenly. For instance, if there are two faulty nodes in the case where the number of frequency bins is 64 and the number of nodes is 32 as in this experiment, beamforming tasks of the four frequency bins assigned to the two faulty nodes have to be reassigned to other non-faulty nodes. Hence, four non-faulty nodes perform the beamforming tasks for three frequency bins while all the other non-faulty nodes do the same for two frequency bins. However, as the number of faulty nodes increases, the communication time of DFD is slightly decreased because the faulty nodes are not included in the communication. The execution times of MCMVDR-DSD are lower than those of MCMVDR-DFD because DSD provides more balanced workload distribution than DFD when the workloads of the faulty nodes are reassigned into non-faulty nodes. DSD reallocates workloads for all grid points (i.e., 3200 grid points in this case) into all the non-faulty nodes more evenly than DFD since the granularity of a parallel task in DSD is much smaller than DFD.

The execution times of RRMVDR-DFD are much lower than those of the two MCMVDR algorithms because the computational complexity of RRMVDR is not increased as shown in Table 1 as the number of faulty nodes increases. However, as the number of faulty nodes increases from zero to two, the execution time of RRMVDR-DFD is rapidly increased as in MCMVDR-DFD due to the imbalanced workload distribution in DFD. In addition, as the number of faulty nodes increases from two to eight in RRMVDR-DFD, the parallel execution times stay about the same as those of the others since their workload imbalance situations are the same as in these faulty-node configurations. The execution times of RRMVDR-DSD are slightly higher than those of RRMVR-DFD because the execution time of the ED stage in RRMVDR-DSD involves a large portion to the overall parallel execution time since the ED stage is not parallelized. Furthermore, as the number of faulty nodes increases, the parallel execution times of RRMVDR-DSD are slightly increased. The reason is that the workloads assigned to all the non-faulty nodes are increased in DSD since the faulty nodes are excluded from parallel processing in the distributed array system.

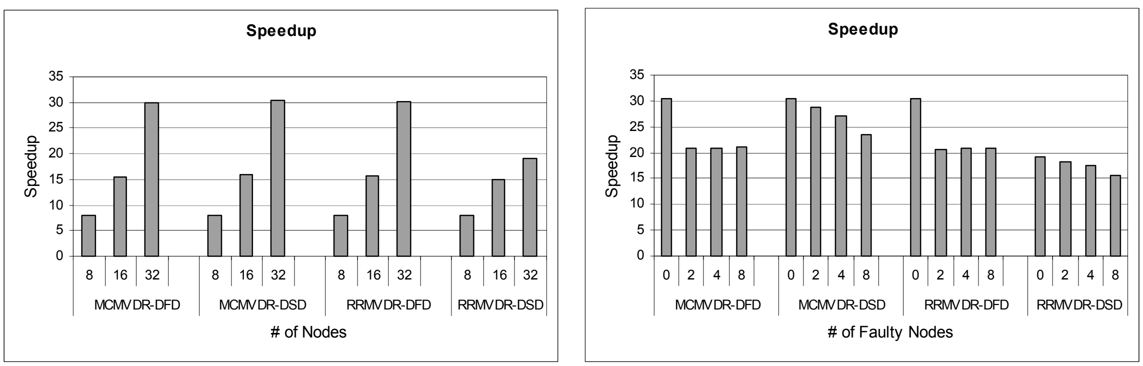
6.4. Speedup

Fig. 9a illustrates the scaled speedup of the fault-tolerant parallel algorithms measured on the PC cluster. Scaled speedup is defined as the ratio of sequential execution time to parallel execution time. This performance metric takes into account the fact that the problem size is also increased as the number of processing nodes is increased, since each node has its own sensor. All the fault-tolerant parallel algorithms except RRMVDR-DSD exhibit promising scaled speedup as system size increases. The speedup of RRMVDR-DSD does not scale as well with system size since the sequential ED stage considerably increases the parallel execution time as discussed in Section 4. The MCMVDR-DSD algorithm shows marginally better scaled speedup than the other algorithms because DSD exhibits a lower communication overhead than DFD due to its small message size and the sequential bottleneck of the INV stage in MCMVDR is lower than that of the ED stage in RRMVDR.

The speedup of the fault-tolerant parallel algorithms is examined in Fig. 9b where the number of faulty nodes varies from zero to eight and system size is fixed at 32 nodes. From the results, as expected, it can be seen that speedup of all the fault-tolerant algorithms decreases as the number of faulty nodes increases because the faulty nodes are excluded from parallel processing thereby causing an imbalance in workload distribution in DFD. The other reasons are that the computational load of MCMVDR is increased and the overall processing power of the distributed array system is naturally decreased with the fixed system size as the number of faulty nodes increases. For instance, speedup of DFD abruptly decreases as the number of faulty nodes varies from zero to two due to its severe imbalanced workload assignment as described in Section 6.3. However, as the number of faulty nodes increases from two to eight in DFD, the speedup stays about the same as that of the others since their workload imbalance situations are the same as in these faulty-node configurations.

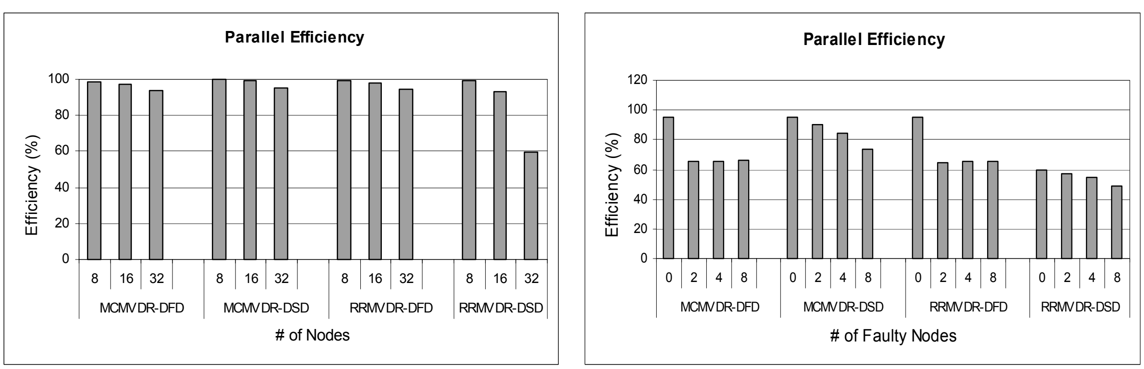
6.5. Parallel efficiency

Parallel efficiency is the ratio of speedup over ideal speedup, which is equal to the number of processors. This performance metric is employed to examine the average utilization of all processors throughout the entire execution of parallel programs. The parallel efficiency versus the number of nodes with no faulty node is shown in Fig. 10a. Fig. 10b illustrates the parallel efficiency versus the number of faulty nodes with a fixed system size of 32 nodes. With no faulty nodes, all the fault-tolerant parallel algorithms except RRMVDR-DSD achieve a promising parallel efficiency between 93% and 99% for the given system. However, as shown in Fig. 10b, the four fault-tolerant parallel algorithms exhibit lower parallel efficiency than the no-faulty-node case because there is a workload imbalance between the non-faulty nodes in DFD. The computational load of MCMVDR is increased and the processing power of the distributed array systems is reduced when the number of faulty nodes is increased. The MCMVDR-DSD algorithm achieves marginally better parallel efficiency than other algorithms since it provides better balanced workload distribution and efficient communication when there are faulty nodes.



(a) (b)

Fig. 9. Speedup versus system size (a) and number of faulty nodes (b).



(a) (b)

Fig. 10. Parallel efficiency versus system size (a) and number of faulty nodes (b).

7. Conclusions

For real-time sonar systems operating in severe underwater environments, high-performance and fault-tolerant parallel processing techniques are required in order to meet real-time and high reliability requirements in the presence of processor and sensor failures. In this paper, four efficient fault-tolerant parallel algorithms are presented to tolerate node failures in distributed sonar array systems with graceful degradation. The four fault-tolerant parallel algorithms are based on dynamic domain decomposition methods and achieve low overhead, minimum redundancy, and fast recovery time for adaptive MFP algorithms in distributed array systems. The performance of the fault-tolerant parallel algorithms is investigated in terms of execution time, speedup, and parallel efficiency on a distributed testbed.

The parallel fault-tolerant algorithms achieve promising parallel performance with low overhead by using an application-level fault tolerance method for the given system sizes and a varying number of faulty nodes in a fixed system size. The experimental results demonstrate that, with no faulty nodes, all the fault-tolerant parallel algorithms except RRMVDR-DSD achieve a promising parallel efficiency between 93% and 99% for given system sizes. However, the four fault-tolerant parallel algorithms with faulty nodes obtain lower parallel performance than with no faulty node because there is a workload imbalance between processing nodes in DFD and the computational load of MCMVDR is increased as the number of faulty nodes increases. Naturally, the overall processing power of the distributed array system is decreased with the fixed system size as the number of faulty nodes increases. The MCMVDR-DSD algorithm achieves marginally better parallel efficiency than the other algorithms since it provides more balanced workload distribution and efficient communication load. Given the increasing demands for high-performance and dependability in distributed sonar systems, these fault-tolerant parallel algorithms can provide real-time, lightweight, and fault-tolerant implementations for adaptive MFP algorithms on distributed sonar array systems.

For future research, the fault-tolerant parallel algorithms presented in this paper can be extended to more enhanced and sophisticated forms of beamforming algorithms to tolerate node failures. The fault-tolerant parallel algorithms need to be integrated with a fault detection method not included in this research to provide more reliable fault-tolerant implementations in distributed sonar array systems.

Acknowledgements

We acknowledge and appreciate the support provided by the Office of Naval Research on Grant N00014-99-10278. Special thanks go to Byung Il Koh in the HCS Lab at the University of Florida for his assistance with useful suggestions.

References

1. W. Liu and V. Prasanna, "Utilizing the power of high-performance computing," *IEEE Signal Processing Magazine* **15** (5), 85-100 (1998).
2. W. A. Rosen and A. D. George, "Fault tolerance in autonomous acoustic arrays," *J. of Franklin Institute* **336B**, 19-32 (1999).
3 P.D. Hough, M. E. Goldsby, and E. J. Walsh, "Algorithm-dependent fault tolerance for distributed computing," Sandia Report SAN2000-8219 (2000).
4. M. Vijay and R. Mittal, "Algorithm-based fault tolerance: a review," *Microprocessors and Microsystems* **21**, 151-161 (1997).

5. C. J. Anfinson, A. W. Bojanczyk, F. T. Luk, and E. K. Torng, "Algorithm-based fault tolerant techniques for MVDR beamforming," *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2417-2420 (1989).
6. G. Bosilca, A. Bouteiller, F. Cappello, S. Djilali, G. Fedak, C. Germain, T. Herault, P. Lemarinier, O. Lodygensky, F. Magniette, V. Neri, and A. Selikhov, "MPICH-V: Toward a scalable fault tolerant MPI for volatile nodes," *Proc. ACM/IEEE Conference on Supercomputing*, 1-18 (2002).
7. J. Haines, V. Lakamraju, I. Koren, and C.M. Krishna, "Application-level fault tolerance as a complement to system-level fault tolerance," *J. Supercomputing* **16**, 53-68 (2000).
8. A. George and K. Kim, "Parallel algorithms for split-aperture conventional beamforming," *J. of Computational Acoustics* **7** (4), 225-244 (1999).
9. A. George, J. Garcia, K. Kim, and P. Sinha, "Distributed parallel processing techniques for adaptive sonar beamforming," *J. of Computational Acoustics* **10** (1), 1-23 (2002).
10. P. Sinha, A. George, and K. Kim, "Parallel algorithms for robust broadband MVDR beamforming," *J. of Computational Acoustics* **10** (1), 69-96 (2002).
11. K. Kim, "Parallel algorithms for distributed in-situ array beamforming," Ph. D. Dissertation, Univ. of Florida, Gainesville, FL (2001).
12. K. Cho, A. George, R. Subramaniyan, and K. Kim, "Parallel algorithms for adaptive matched-field processing on distributed array systems," *J. of Computational Acoustics*, **12** (2), 149-174 (2004).
13. K. Cho, "Parallel and fault-tolerant techniques for adaptive matched-field processing on distributed array systems," Ph. D. Dissertation, Univ. of Florida, Gainesville, FL (2004).
14. A. B. Baggeroer, W. A. Kuperman, and H. Schmidt, "Matched-field processing: Source localization in correlated noise as an optimum parameter estimation problem," *J. Acoust. Soc. Am.* **83** (2), 571-587 (1988).
15. H. Schmidt and A. B. Baggeroer, "Environmentally tolerant beamforming for high resolution matched field processing: deterministic mismatch," *J. Acoust. Soc. Am.* **88** (4), 1851-1862 (1990).
16. J. M. Ozard, M. J. Wilmut, D. G. Berryman, and P. Z. Zakarauskas, "Speed-up performance of a nearest-neighbors algorithm for matched-field processing with a vertical line array," *Proc. Oceans '93 Engineering in Harmony with Ocean*, 86-90 (1993).
17. W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, "Numerical recipes in C," Cambridge University Press, New York (1992).
18. M. Porter, "The KRAKEN normal-mode program," SCALANT Memorandum, SM-245 (1991).
19. A. Tolstoy, "Matched field processing for underwater acoustics," World Scientific, Singapore (1993).
20. A. B. Baggeroer, W. A. Kuperman, and P. N. Mikhalevsky, "An overview of matched field methods in ocean acoustics," *J. Acoust. Soc. Am.* **18** (4), 401-423 (1993).
21. D. K. Paradhan, "Fault-tolerant computer system design," Prentice Hall, New Jersey (1996).
22. D. K. Paradhan and N. H. Vaidya, "Roll-forward checkpointing scheme: A novel fault-tolerant architecture," *IEEE Trans. on Computers*, **43** (10), 1163-1174 (1994).
23. N. Lee, L. M. Zurk, and J. Ward, "Evaluation of reduced-rank, adaptive matched-field processing algorithms for passive sonar detection in a shallow-water environment," *Thirty-Third Asilomar Conference on Signals, Systems, and Computers* **2**, 24-27 (1999).
24. G. B. Smith, C. Feuillade, and D. R. Delbalzo, "Macthed-field processing enhancement in a shallow-water environment by incoherent broadband averaging," *J. Acoust. Soc. Am.* **91** (3), 1447-1455 (1992).
25. J. M. Ozard and G. H. Brooke, "Improving performance for matched-field processing with a minimum variance beamformer," *J. Acoust. Soc. Am*. **91** (1), 141-150 (1992).